

FIG. 1

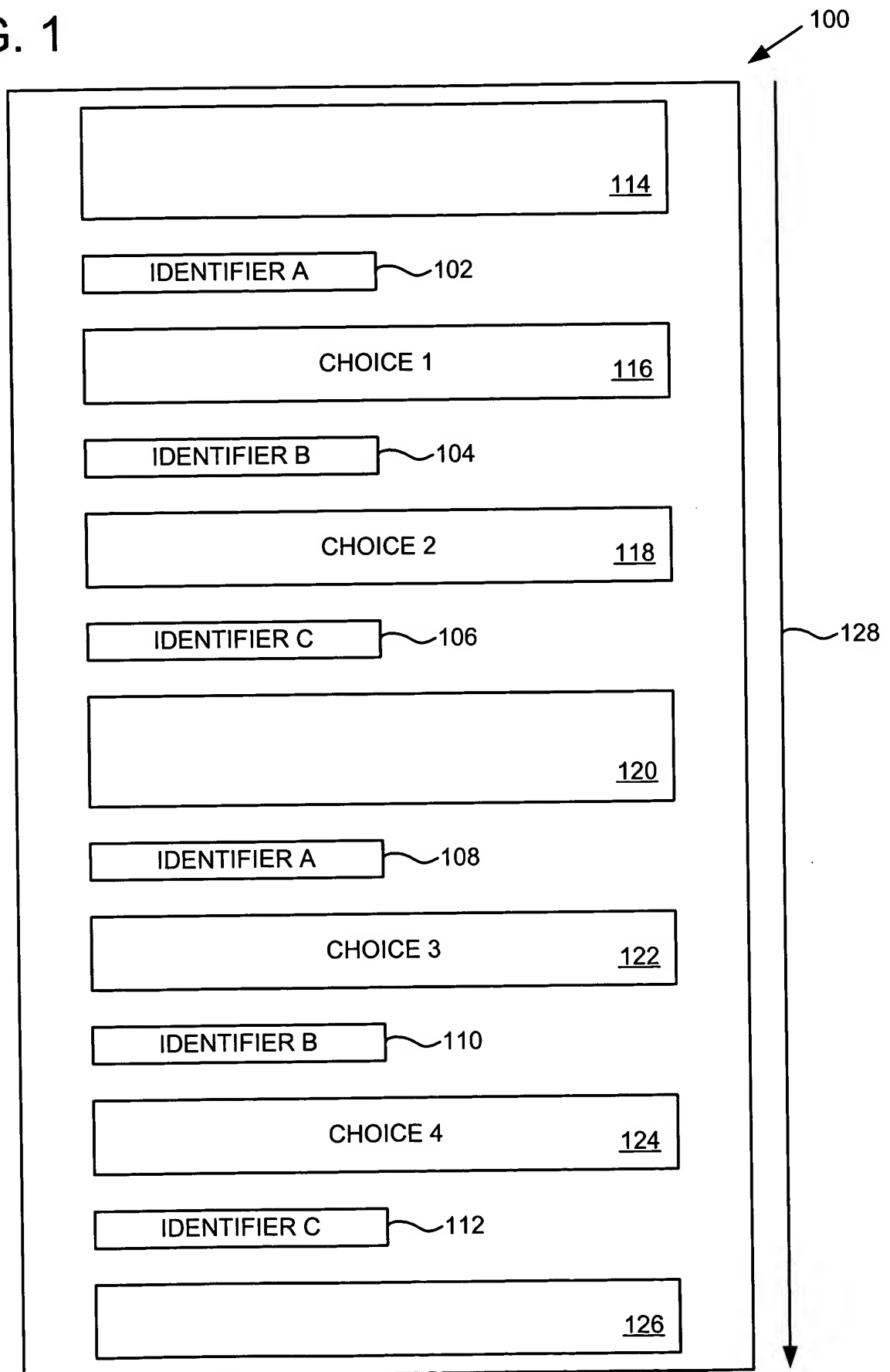




FIG. 3

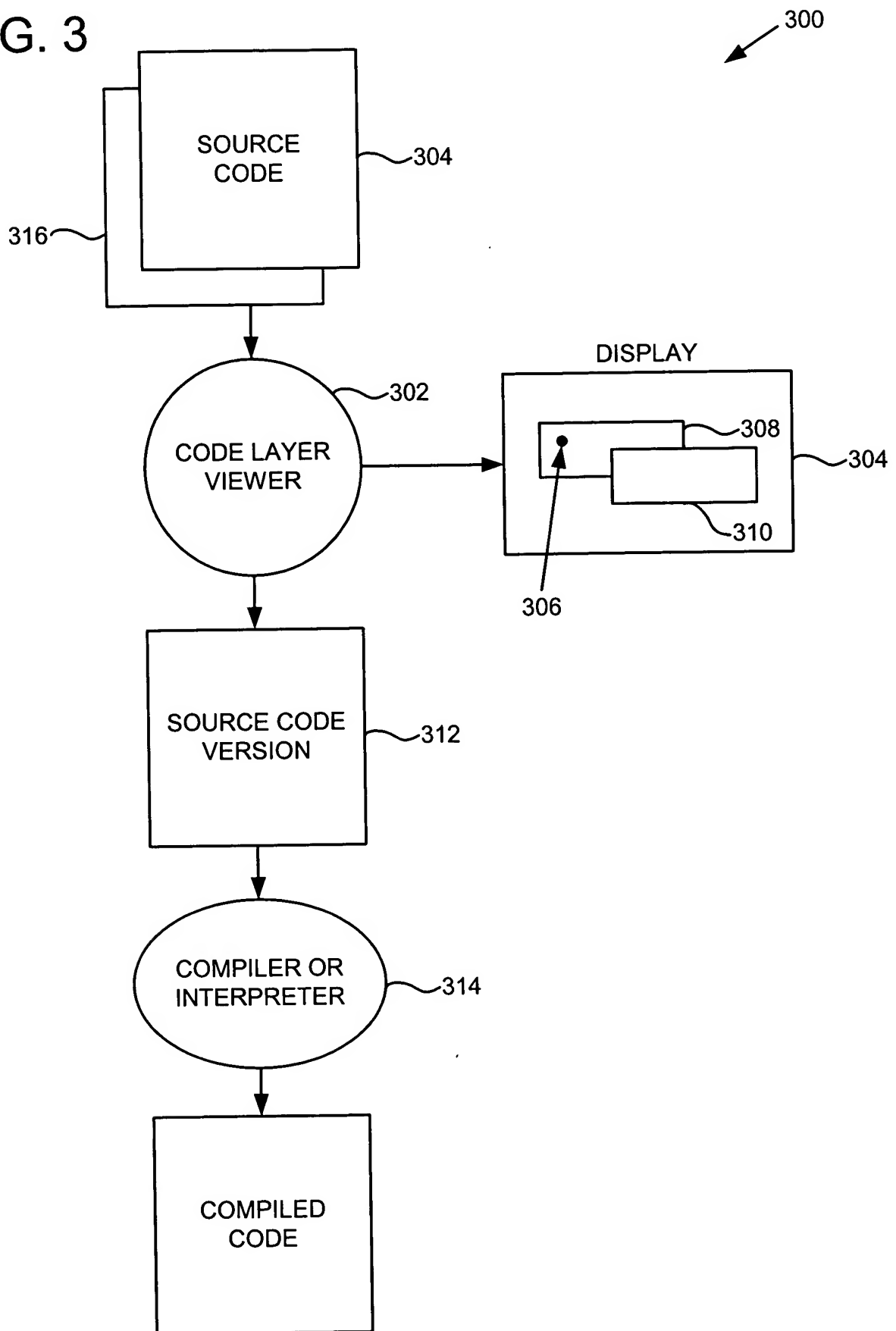


FIG. 4

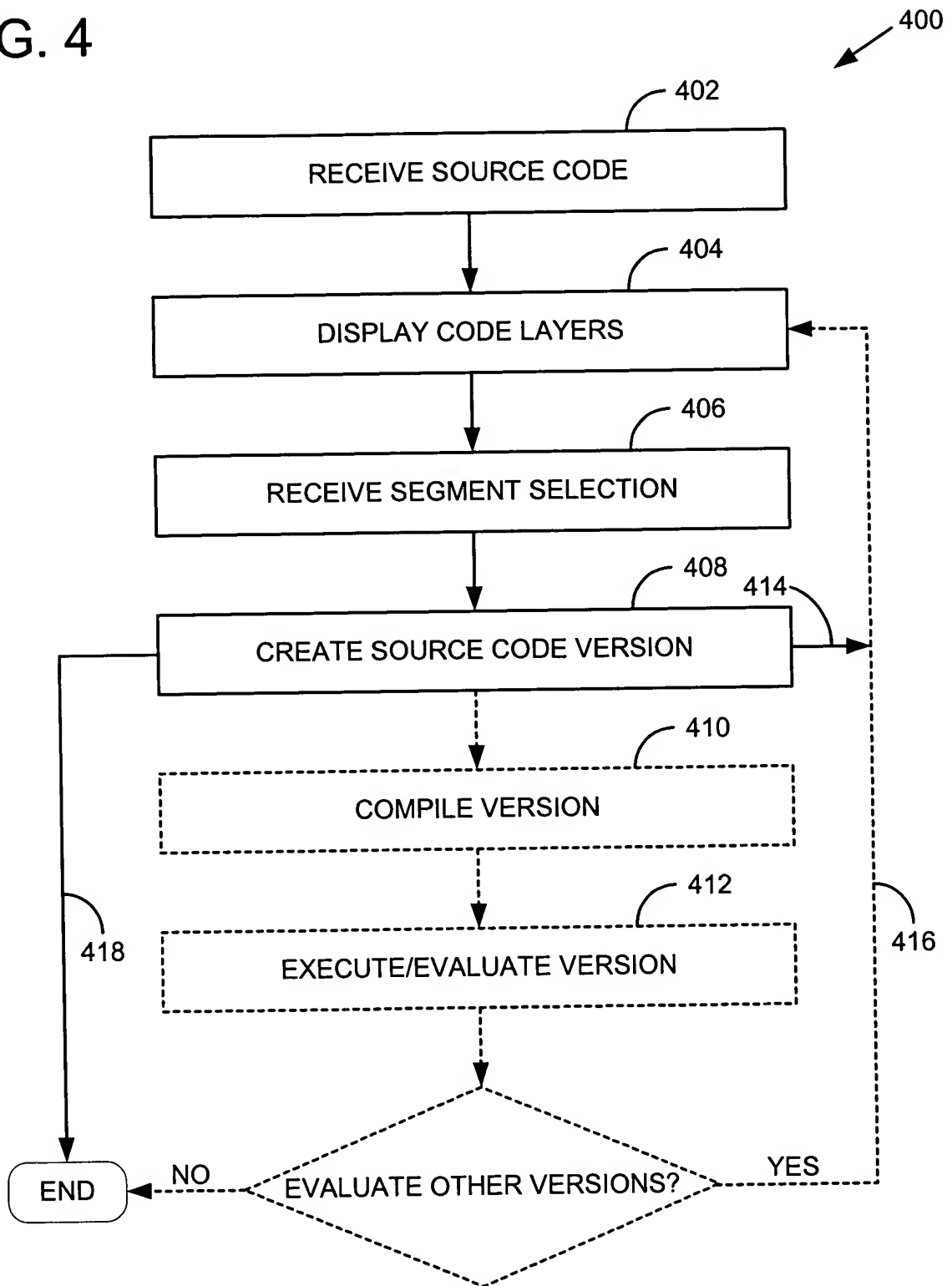


FIG. 5

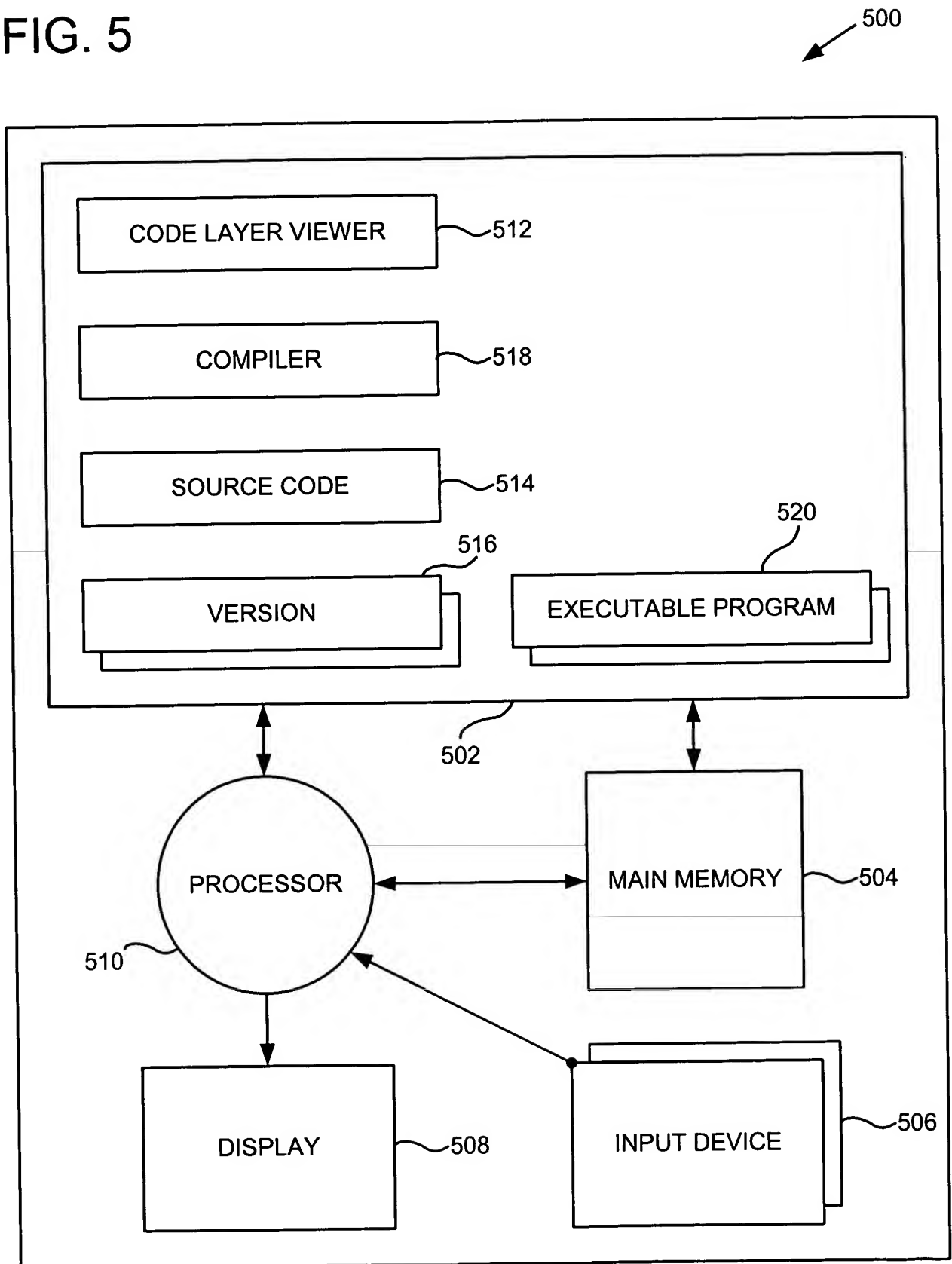


FIG. 6

600 →

```
class CodeNode ← 602

class SimpleCodeNode extends CodeNode ← 604
  code as String ← 606

class HighLowCodeNode extends CodeNode ← 608
  highNodes as Seq of CodeNode ← 610
  lowNodes as Seq of CodeNode ← 612
```

FIG. 7

700 →

```
class Source
  codeNodes as Seq of CodeNode ← 702
  codeLayer as CodeL ← 704
```

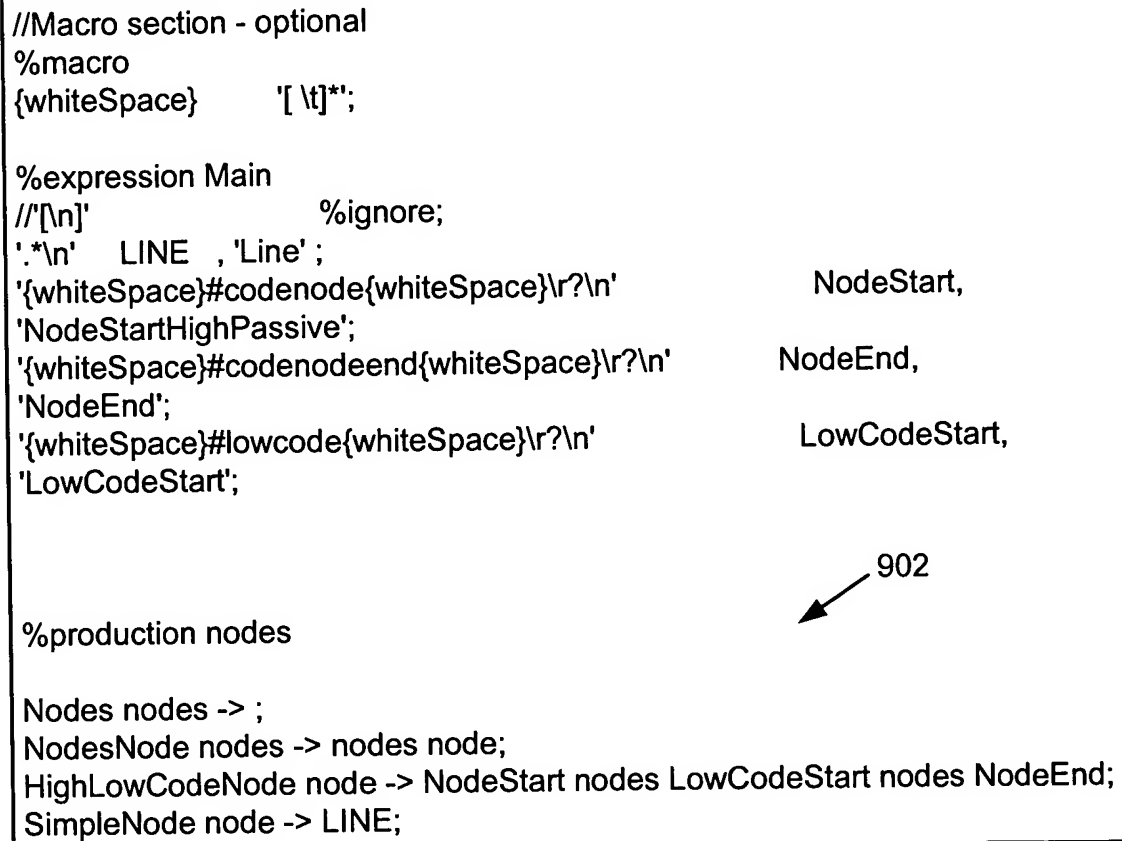
FIG. 8

800 →

```
class Source
  ToString(source as Source, codeLayer as CodeLayer) as String
    var s as String=""
    step foreach node in source.codeNodes
      s+=ToString(node,codeLayer) ← 804
    step
    return s
    ← 802

  ToString(codeNode as CodeNode ,codeLayer as CodeLayer) as String
    match codeNode
    simpleNode as SimpleCodeNode: ← 808
      return simpleNode.code
    node as HighLowCodeNode: ← 810
      let activeNodes=if node in codeLayer.highNodes
        node.highNodes ← 812
      else
        node.lowNodes ← 812
      var s as String=""
      step foreach node in activeNodeSeq
        s+=ToString(node,codeLayer) ← 814
      step
      return s
```

## FIG. 9



```
//Macro section - optional
%macro
{whiteSpace}      '[ \t]*';

%expression Main
//[ \n]'           %ignore;
'.*\n'   LINE    , 'Line' ;
'{whiteSpace}#codenode{whiteSpace}\r?\n'      NodeStart,
'NodeStartHighPassive';
'{whiteSpace}#codenodeend{whiteSpace}\r?\n'    NodeEnd,
'NodeEnd';
'{whiteSpace}#lowcode{whiteSpace}\r?\n'        LowCodeStart,
'LowCodeStart';

%production nodes

Nodes nodes -> ;
NodesNode nodes -> nodes node;
HighLowCodeNode node -> NodeStart nodes LowCodeStart nodes NodeEnd;
SimpleNode node -> LINE;
```

FIG. 10

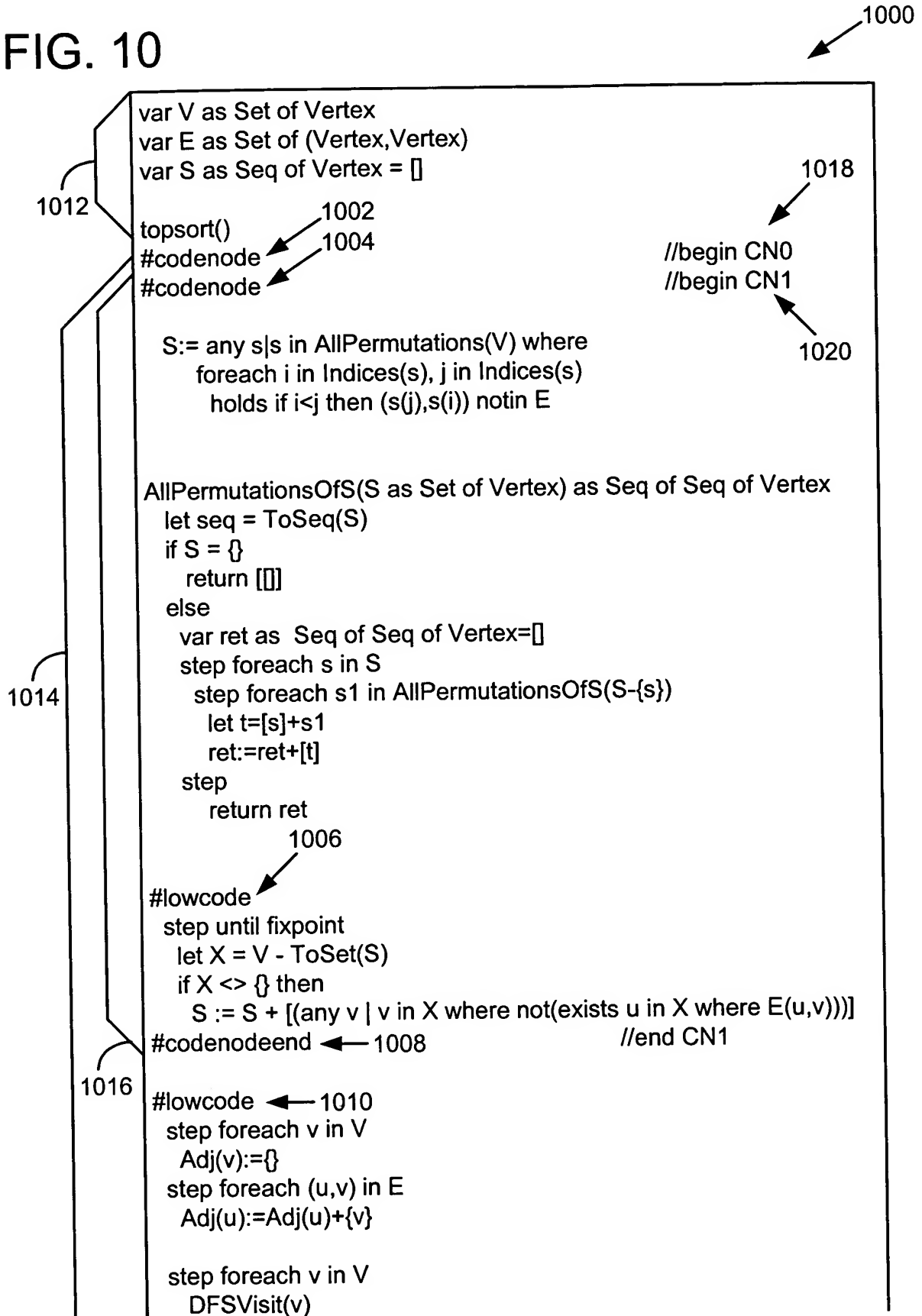




FIG. 11

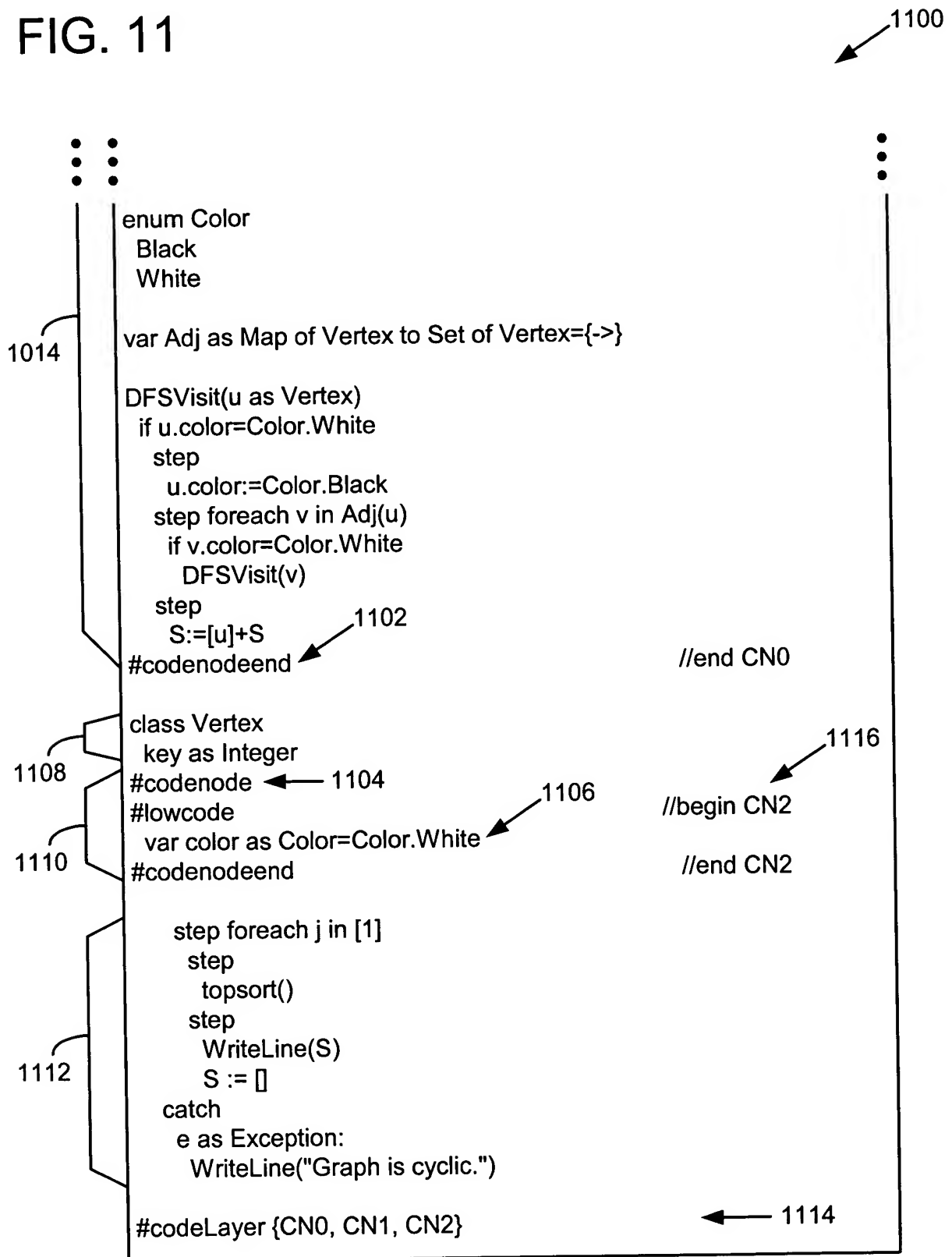


FIG. 12

1200 →

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex) 1202
var S as Seq of Vertex = []

topsort()
  S:= any s|s in AllPermutations(V) where
    foreach i in Indices(s), j in Indices(s)
      holds if i<j then (s(j),s(i)) notin E

AllPermutationsOfS(S as Set of Vertex) as Seq of Seq of Vertex
  let seq = ToSeq(S)
  if S = {}
    return [[]]
  else
    var ret as Seq of Seq of Vertex=[]
    step foreach s in S
      step foreach s1 in AllPermutationsOfS(S-{s})
        let t=[s]+s1
        ret:=ret+[t]
    step
    return ret

class Vertex 1204
  key as Integer

step foreach j in [1]
  step
  topsort()
  step
  WriteLine(S)
  S := []
catch
  e as Exception:
    WriteLine("Graph is cyclic.") 1206
```

FIG. 13

1300 →

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()

step until fixpoint
  let X = V - ToSet(S)
  if X <> {} then
    S := S + [(any v | v in X where not(exists u in X where E(u,v)))]

class Vertex
  key as Integer

step foreach j in [1]
  step
    topsort()
  step
    WriteLine(S)
  S := []
catch
  e as Exception:
    WriteLine("Graph is cyclic.")
```

FIG. 14

1400

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()
  step foreach v in V
    Adj(v):={}
  step foreach (u,v) in E
    Adj(u):=Adj(u)+{v}

  step foreach v in V
    DFSVisit(v)

enum Color
  Black
  White

var Adj as Map of Vertex to Set of Vertex={->}

DFSVisit(u as Vertex)
  if u.color=Color.White
    step
      u.color:=Color.Black
    step foreach v in Adj(u)
      if v.color=Color.White
        DFSVisit(v)
    step
      S:=[u]+S

class Vertex
  key as Integer
  var color as Color=Color.White

  step foreach j in [1]
    step
      topsort()
    step
      WriteLine(S)
      S := []
  catch
    e as Exception:
      WriteLine("Graph is cyclic.")
```

FIG. 15

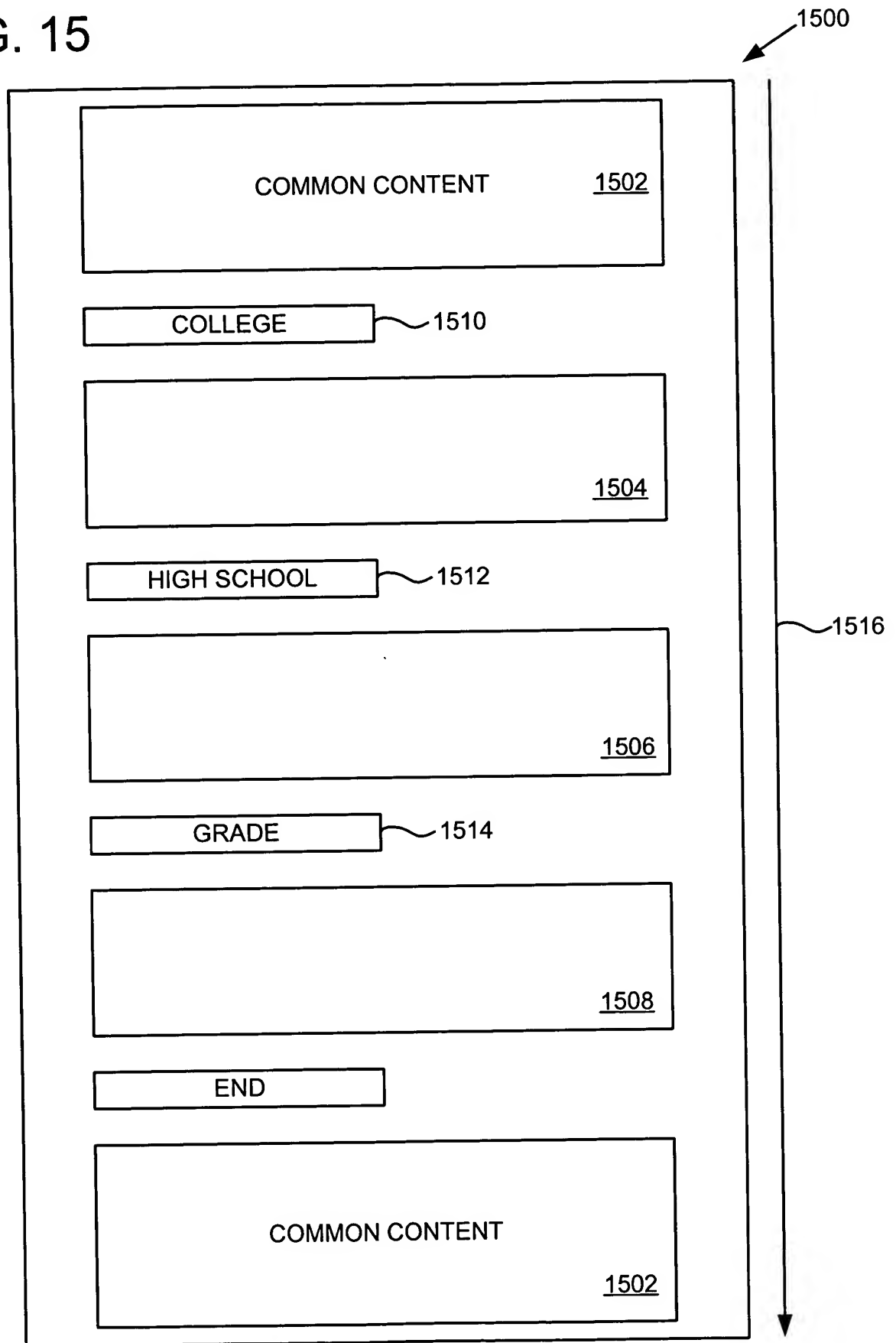


FIG. 16

